The Web Developer Bootcamp

by Colt Steele on Udemy

Notes

by E.Ömer EROL

https://github.com/eomereu/webdevbootcamp
https://www.udemy.com/course/the-web-developer-bootcamp/

Chapter 1
Introduction to HTML & Intermediate HTML

- Hitting Tab after typing html in Sublime, creates the boilerplate automatically for us. (After giving the file .html extension)

- MDN: Mozilla Developer Network (https://developer.mozilla.org/en-US/)

- `<html>` tab is the root of an HTML file, it is followed by `<head>` and `<body>` tags

- `<head>` tag, provides general information; so basically everything that we don't see on the paper as a user, things like stylesheet files, js files; they all go in the head. In conclusion, here is where we put metadata

- `<body>` is where we put all our content

- <*!-- This is a comment --*>, simply to fastly create a comment, select the content or go on a line and hit 'Ctrl + /'

- Although `<title>` won't show up on our page, it's important because it is the text that goes up to the tab name. And pay attention to the reason it won't show up on the page is it is in the head/metadata section. So it is also the name that shows up on the google results

- MDN Element Reference: Lists all the elements of HTML (https://developer.mozilla.org/en-US/docs/Web/HTML/Element) around 100 or so but 15-20 of'em frequently used

- `<h1>` is the most, `<h6>` is the least important heading... Tags like this headings called as 'block level elements' which basically means even we write'em on a single line they are seperately shown on lines on the webpage

- 'Inline elements' are the ones which if written on the same line they are shown on the same line.

- `<p>` Paragraf tag makes the text block level

- `<b>` tag is the basic way to bolden a text; this is inline also; but there is a better tag to do it which is `<strong>` tag. Although it does exactly the same thing it emphasizes the importance of it in the source code -as far as I guess- and things like these tags called 'semantic mark' in HTML5

- `<i>` makes the text italic, it's inline; like the above one, `<em>` tag makes the same but a better way in terms of semantic marks

- `<ol>` tag creates an ordered list, which means every item takes number in front of it as 1,2,3 etc, and inside this list we have to specify every one of the elements with `<li>` tags

- `<ul>` tag creates an unordered list, which just takes bulletpoints in front of'em, again inside this list we have to specify every one of the elements with `<li>` tags

- we can make lists in lists

- Ctrl + Shift + D, automatically copies the current line and pastes on next line directly.

- First headings, outer lists, inner lists

- Inside an html file, typing "lorem" and hitting Tab right after it, is going to give a long text to us, automatically.

- `<div>` is a way of grouping things together. But actually it won't give us much benefit until we get to CSS. Because then we will go and style that group together. It is a block level element btw, which means inner divs will cause new lines

- `<span>` is also a generic container but there is a key difference is an inline one. So we will use this one inside of a div if we want to give that part a more specific feature along with the div features.

- Attributes: adding additional information to tags. Its form is like the following, a key-value pair,

```
<tag name="value"></tag>
```

- `<br>` tag is a break line tag, which passes you to the next line

- We don't need a closing tag for `<img>`, `<br>` attributes.

- Anchor Tag is the one we all see in every Google search result actually, usage:

```
<a href="url">Link Text</a>
```

I need to make my link explicit at this point. So I have to add that "http://" part at the beginning otherwise it tries to find that following part at my current folder!

- Table usage:

```
<table>
 <thead>
  <tr>
   <th>Heading for 1st column</th>
   <th>Heading for 2nd column</th>
  </tr>
 </thead>
 <tbody>
  <tr>
   <td>1st row 1st column</td>
   <td>1st row 2nd column</td>
  </tr>
 </tbody>
</table>
```

tr used for creating rows, td for columns, th for headings

`<table border="1">` can be used to determine the borders but it makes more sense to do it with CSS

`<thead>` and `<tbody>` tags are like the head and body tags of the page, it is optional but better to use

`<img width="">` can be used for reararranging the size of an image but actually we will be doing this by CSS later on. PS, it automatically scales the image by the given width

- In order to create multiple cursors in Sublime, simply hold Ctrl and click on places that we want the cursor to be on

- HTML Forms: Getting input from user:

- `<form></form>`: It's a shell/container for inputs like username, mail, password, log in button, dropdown menus, radio buttons etc. So by just forming a 'form' we don't actually create something visual, all buttons etc. must be written explicitly in this for something visual.

The following attribute pair sends data from the form to a server somewhere. Action atr contains the URL to send form data to and Method specifies the type of the HTTP request. We use 'get' request when we try to retrieve some data like searching etc. and we use 'post' request when we are sending data like wanting it to be added to a database or posted to a server. i.e. Google search is get request; Facebook signup is post request:

```
<form action="/my-form-submitting-page" method="post"> ALL THE INPUT </form>
```

By default if we don't specify the URL part it is the current location. If we don't specify the method it's a 'get' request by default.

- `<input>` tag creates interactive controls.
The "type" atr determines the types of input.

```
<input type="text/date/color/file/checkbox/password/radio/...">
```

The "name" atr specifies the key of the current input in the querystring.

```
<input name="username" type="username">
```

The "placeholder" atr is the default text inside a box

```
<input name="username" type="username" placeholder="username">
```

- Ctrl + L select the current line

- `<label></label>`: We put `<input>` tags into this to give them a label and make them more comprehensive:

```
<label>
  Username:
  <input name="username" type="username" placeholder="username">
</label>
```

An alternative and a better way to use `<label>` tag with `<input>` tags is as the following, by using "for" and "id" ATRs. By doing so we prevent nests:

```
<label for="username">Username:</label>
<input id="username" type="text" placeholder="username">
```

In order to add some validation, we can first use to check the presence of some input, "required" label. When we change type to "email" for example, it will automatically inspect the input is appropppriate for an email or not.

```
<input id="email" name="email" type="email" placeholder="email" required>
```

- Radio Boxes:

Regarding radio boxes, actually we can select more than one in radio button but cannot unselect it so we have to arrange it manually to make exactly one choice, to do this we give the choices the same "name"!, but here we must also specify the "value" of the choice for a better info sending and a better query; Used in things like gender selection etc. (Don't forget to add Labels)

```
<input id="male" name="gender" type="radio" value="male">
<input id="female" name="gender" type="radio" value="female">
```

- `<input type="checkbox">`

A checkbox is the box which we can make a tick or not like selecting our skills, multiple or none

- A `<button></button>` at the end of a form will simply submit the form

- Select Tag:

With this tag we can create a nice dropdown selection menu. Usage:

```
<select name="color">
  <option>Red</option>
</select>
```

Here because we didn't specify a value, it will automatically set the value in the query to the string on the option which is "Red" here. But sometimes we may want to specify it explicitly, to do so:

```
<select name="mood">
  <option value="Happy">:)</option>
</select>
```

Now the value will be set to "Happy" in the query if we select ":)"

- Textarea Tag:

This tag is used for inputs that are more than one line. "rows" & "cols" ATRs can be used to specify the size of the box,

- Pattern & Required Title ATRs,

```
<input id="password" type="password" name="password" pattern=".{5,10}" required title="errorMsg">
```

We can type as much or as less as we want but cannot submit due to the pattern and the title (by adding "Please match the requested format" on the above line of our title) will be shown.

But if we use the following,

```
<input id="password" type="password" name="password" minlength="5" maxlength="10" required title="errorMsg">
```

Here it restricts the input between 5 to 10. We cannot even type nor less than 5 neither more than 10 characters. And the title won't be showed up to the user.

Chapter 2
Introduction to CSS / CSS-1

CSS (Cascading Style Sheets)
- They are actually seperate files that we include into our HTML files.

- The General Rule:
  that every line we write will follow...

```
selector {
  property: value;
  anotherProperty: value;
}
```

- 'Inline' styling is not a good idea because it doesn't seperate HTML and CSS files. Furthermore it's a lot of work to form lots of things all one at a time and especially when we want to make some changes to multiple elements,

```
<h3 style="color: pink;">My Heading</h3>
```

- 'Style Tag' is also a bad idea, -on the other hand generally used for quick demonstrations and checks- coz it's also included in the HTML file but as a seperate part on top in 'head' part; which is mainly allows us to write some CSS code inside this tag,

```
<style type="text/css">
  .
</style>
```

PS: / Comment in CSS */*

- So finally to write our CSS in a different file, we use <link> tag inside the HTML file. Hitting tab after typing 'link' will create a nicely prepared tag for us.,

```
<link rel="stylesheet" type="text/css" href="CSSFileName.css">
```

PS: It must be included in the 'head' section!

- There are three types of color system:
  1.Hexadecimal
  2.RGB
  3.RGBA
  Actually the former two refer to the same thing but have differences in both syntax and base selection. The differencelatter two

- Hexadecimal colours contain six digits following a '#'. It's using base 16.
  `#_ _ _ _ _ _` : First 2 represents how much red is the colour and the others green, blue respectively.
  red grn blu

```
#FFFFFF : White
#000000 : Black
```

- RGB colours have slight different showing style. It's using base 10, the values show red, green, blue

```
rgb(_, _, _)
     r  g  b
rgb(255,255,255): White
rgb(0,0,0): Black
```

- RGBA system adds a fouth field as transparency value whose range is between 0 and 1, (A stands for Alpha)

```
rgba(_, _, _, ._)
      r   g  b   t
```

`rgba(100,82,12,1)`: No transparency
`rgba(100,82,12,0)`: Fully transparent (nothing shows up)
`rgba(100,82,12,.6)`: Little transparency (as getting closer to 1)
`rgba(100,82,12,.3)`: More transparency (as getting closer to 0)

- Background Property Usage:

```
   background: rgb(255,255,255); /* full color */
   background:
url(https://badasshelmetstore.com/wp-content/uploads/2016/10/avengers-
helmets.jpg); /* image */
```

Alone this will replicate the image all along the page so be careful to pick a continous one if using this.

- Some Background Properties:

```
   background-repeat: no-repeat; /* If we don't want the image to be repeated as above we use this
*/
   background-size: cover; /* If we don't use this property the page won't cover the page. It won't
strecth out or won't fit depending on the resolution; if we don't use this */
```

- Border Usage:

Normally we use border with 3 must-have properties:
1. width
2. style
3. color

So the original code looks like this:

```
   border-width: 3px;
   border-style: solid;
   border-color: white;
```

But we use the following in terms of being practical:

```
   border: 3px solid white; /* widht style color respectively */
```

CSS Selectors:
1.Element
2.ID
3.Class

1.Element Selector:

Selects all instances of a given element.

```
   div {
      background: aquamarine;
   }
```

2.ID

Selects an element with a given ID. IDs have to be unique! It's not possible to have multiple instances with the same ID! We use hashtag '#' while referring to IDs.

```
   <p id="mytext">Hallo!</p>
   -------------------------
   #mytext {
      background-color: yellow;
   }
```

3.Class

Selects all elements with the given class. We use dot '.' while referring to classes.

```
   <p class="important">Achtung!</p>
   <p class="important">Das ist verboten!</p>
   ----------------------------------------
   .important {
      color: red;
   }
```

- To load the checkboxes as checked while loading the page, we add 'checked' ATR in the input tag of the checkbox,

```
   <input type="checkbox" checked>
   Fertig!
```

- Google Chrome Inspect Element:

We can right click while on a webpage to inspect element. It will allow us to see both the HTML and the CSS part of it. So we can use this feature both for testing and trying things easily while creating some things and for inspecting stylings of other webpages and replicating them.

If we right click on a specific element and then click on inspect element it will directly show us that specific element within the opened up code segment.

Here on the right side, 'Styles' tab will consist the CSS stylings! We can directly on and off things or manipulate things here and watch the changes.

On the left side we can click on 'Magnifier' icon to go to sources of some things on the page by then clicking on'em.

More Selectors:

*/*Element*/* -> Selects all elements of the defined type

```
li {

}
```

*/*class*/* -> Selects all elements with the specified class

```
.hello {

}
```

*/*id*/* -> Selects an element with a specific and unique ID

```
#name {

}
```

*/*Star*/* -> Selects everything on a page

```
* {
   border: 1px solid lightgrey;
}
```

*/*Descendant Selector*/* -> Selects element(s) inside an element(s)

```
ul li a {
   color: red;
}
```

*/*Adjacent Selector*/* -> Selects an element after an element.

```
h4 + ul {
   border: 4px solid red;
}
```

*/*Attribute Selector*/* -> Selects all of that ATRs

```
a[href="http://www.google.com"] {
   background: blue;
}
```

*/*nth of type*/* -> Selects nth element of every odd/even number of that element

```
li:nth-of-type(3){
   background: purple;
}

li:nth-of-type(odd){
   background: purple;
}
```

Inheritence & Specificity

'Inheritence' is the traditional thing that we know from almost all languages, when we specify,

```
body {
   color: red;
}
```

Everything on the page, inherits this and turns into blue although we don't mark'em; if we want them to be styled differently we gotta write'em explicitly, like,

```
body {
   color: red;
}
ul {
   color: blue;
}
li:nth-of-type(3){
   color: green;
}
```

And here 'Specifity' moves in. It is multiple stylings targeting one element; just like above body, ul and li targets the 3rd li(s) on the page. And in this war, the closest one (in terms of containing) -the most specific one in other words- wins this war.

At this point when we inspect element on a page and look at the styles, we will see that those defeated inherited styles are overlined! And these are showed from bottom to top as more general to more specific one.

Specificity Order: From least to most,

1. */*Type Selectors*/*

```
li {

}

li a {

}

h4 + ul {

}
```

2. */*Class, Attribute & Pseudo-Class Selectors*/*

```
.hallo {

}
input[type="text"] {

}
a[href="http://www.google.com"] {

}
a:hover
input:checked
```

3. */*ID Selectors*/*

```
#hello {

}
```

MORE SPECIFIC SELECTORS!

*/*Make all "checked" checkboxes have a left margin of 50px(margin-left: 50px)*/*

```
input:checked {
   margin-left: 50px;
}
```

/ Make the <label> elements all UPPERCASE without changing the HTML(definitely look this one up*/*

```
label {
   text-transform: uppercase;
}
```

Other Options: lowercase, capitalize...

*/*Make the first letter of the element with id 'special' green and 100px font size(font-size: 100)*/*

```
#special::first-letter {
   color: green;
   font-size: 100px;
}
```

*/*Make the <h1> element's color change to blue when hovered over */*

```
h1:hover {
   color: blue;
}
```

*/*Make the <a> element's that have been visited gray */*

```
a:visited {
   color: gray;
}
```

Other Options: link, visited, hover, active...

Chapter 3
Intermediate CSS

- */* font-family */*

Decides the font type. Type name must be between quotes if it starts with numbers.

```
h1 {
   font-family: Arial;
}
```

- */* font-size */*

Here 'em' is dynamically changing and multiplying the text that it belongs to, according to the one level outer text of the current text

```
body {
   font-size: 16px;
}

h1 {
   font-size: 2.0em;
}
```

- */* font-weight */*

Decides how bold/light text will be. Some of the font types let you use it with a range of 100-800 (not every value but every hundred value) other than normal, bold etc.

```
p {
   font-weight: 800;
}
h2 {
   font-weight: bold;
}
```

- */* line-height */*

Its value multiplies the default line spacing value which is determined by the font

```
p {
   line-height: 2;
}
```

- */* text-align */*

Specifies the alignment

```
h1 {
   text-align: center;
}
```

- */* text-decoration */*

Used to give text underline effect, linethrough effect etc.

```
p {
   text-decoration: line-through;
}
```

- fonts.google.com

Free fonts that Google provides us; for usage:
https://www.youtube.com/watch?v=Z3JR6mEWEEo&feature=youtu.be

- Box Model

In a document, each element is represented as a rectangular box. In CSS, each of these rectangular boxes is described using the standart box model.

Each box has four edges:

1.margin edge: the space outside the box; kinda within the elements. Usages,

```
margin: 20px; /* Same space from all sides */
margin-top: 40px;
margin: TOP RIGHT BOTTOM LEFT; /* More specified space from any side */
margin: 0 auto 0 auto; /* Centers the element from left and right edges */
margin: 0 auto; /* Specifies top to 0 px & right to auto, so indirectly fully centering from right and left again */
```

2.border edge: the thickness of the border. Usages,

```
border: 2px solid blue;
border-width: 2px;
border-style: solid;
border-color: blue;
```

3.padding edge: the space between border and the content inside of it. Usages,

```
padding: 10px;
padding-left: 20px;
```

4.content edge: the text itself (width & hight). Usages,

```
width: 20px;
width: 60%;
```

The space on the right of the content comes from the number 4, which is content edge -width- but the space on the left comes from number 3, which is padding edge -padding-left-. On the other hand, padding has also right spacing, which means we have two adjustments for the right side of our content.

We can use a percentage on the width to make the website more dynamic and change itself depending on the body (window) size,

```
p {
  width: 50%;
}
```

- An element can have multiple classes at once,

```
<td class="vertical horizontal"></td>
```

So this td is in also vertical class and also horizontal class.

- Normally each div and div-like elements go on different lines, that's because what's defined as default as float property. If we change it as the following, all divs will float from left to right to bottom:

```
div {
  float: left
}
```

When we don't modify float property, by default HTML has one space between adjacent images; but when we modify it, that's gone.

- To go to end and the beginning of a line in Sublime with "Alt+Right"/"Alt+Left"; paste the following into the Key Bindings within Preferences:

```
{ "keys": ["alt+left"], "command": "move_to", "args": {"to": "bol", "extend": false} },
{ "keys": ["alt+right"], "command": "move_to", "args": {"to": "eol", "extend": false} }
```

- It may be more beneficial to have just one h1 -which is the main and the only one- on the page and having the other headings -which are multiple- as h2.

- 'rem' adjusts the text size not according to the size of the parent element like 'em' but according to the root element; which means we don't have to worry about one changing the other,

```
letter-spacing: 0.2rem;
```

Letter spacing -btw- specifies the space between letters

- 'max-width' specifies the max width to be shown as window shrinkes. It makes sense to use it with a percentage width.

```
max-width: 800px;
width: 80%;
```

- `<hr>` tag adds a horizontal line on the page. No closing tags etc.

Chapter 4
Bootstrap 3

Bootstrap:

Bootstrap is a really useful CSS and JavaScript library. It is a single CSS file and a single JavaScript file. Inside the CSS file there is a bunch of staff that we get for free.

Components are the bigger pieces.

We can either download it to our machine and work/load locally (just use link tag for this after moving the 'bootstrap.css' file to the work area) or we can include it online by using,

```
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css"
integrity="sha384-BVYiiSIFeK1dGmJRAkycuHAHRg32OmUcww7on3RYdg4Va+PmSTsz/
K68vbdEjh4u" crossorigin="anonymous">
```

To include Bootstrap 4,

```
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
integrity="sha384-
9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
crossorigin="anonymous">
```

- We can find any info and any choice of any element on the getbootstrap.com,

Bootstrap 3,

https://getbootstrap.com/docs/3.3/css/

https://getbootstrap.com/docs/3.3/components/

Bootstrap 4,

https://getbootstrap.com/docs/4.5/getting-started/introduction/

https://getbootstrap.com/docs/4.5/components/

- To simply overwrite the styles, we can create a 'style' section in the head of our HTML file,

```
<style type="text/css">
   .btn-success {
     background-color: blue;
   }
</style>
```

- Jumbotron

Jumbotron is the big header like thing with an explanation and a button. It takes the 100% of the contain that it is inside of,

https://getbootstrap.com/docs/3.3/components/#jumbotron

https://getbootstrap.com/docs/4.5/components/jumbotron/

- Grid System

To handle the kinda problematic thing above in other words to put some things to the center of the webpage or make it just 1/3 of the page etc. we normally use 'Grid System'. But for a basic solution we can also use the following 'container' class to handle it:

- Container

A container is a predefined class in Bootstrap that used with 'div' tags. It automatically creates a container with predefined useful margines and width etc.

- Forms

The advancedly modified version of the form we have seen back in ChapterOne:

https://getbootstrap.com/docs/3.3/css/#forms

`.form-group | <div class="form-group">`

A label-input pair is better used within "form-group" class to group'em together and add some spacing from the rest.

`.form-control | <input type="email" class="form-control" id="InputEmai" placeholder="email">`

Specifies the style of the input box and its effect when selected.

`.form-inline | <form class="form-inline">`

The class that used with the 'form' tag in order to make the form inline.

`.checkbox | <div class="checkbox">`

When we put label-input(checkbox) pair together inside this div class, the label there turns into a clickable text which is awesome

- We can place 'input' into 'label',

```
<label>
   <input type="checkbox"> Check me out
 </label>
```

- Shit + Tab in Sublime Text takes the selected lines 1 intend backwards.

- Nav Bars

https://getbootstrap.com/docs/3.3/components/#navbar

https://getbootstrap.com/docs/4.5/components/navbar/

In the 'Default Navbar' at first we will realize that the hamburger menu (shows up when we shrink the window) and the 'Dropdown' menus are not working. To make them work we need to include 'JavaScript' bootstrap file, (The following is Bootstrap 3)

```
<script
src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"
integrity="sha384-
Tc5IQib027qvyjSMfHjOMaLkfuWVxZxUPnCJA7l2mCWNIpG9mGCD8wGNIcPD7Txa"
crossorigin="anonymous"></script>
```

And then also include a 'jquery' script over, by copying the most recent/approppriate version and pasting it like we included the Bootstrap JS file with <script> tags,

https://code.jquery.com/

```
<script src="https://code.jquery.com/jquery-3.5.1.min.js"
integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
crossorigin="anonymous"></script>
```

- Tag & classes for creating the navbar,

```
<nav class="navbar navbar-default"></nav>
```

- It's better if we put the things in the navbar into a container div!

-* We put the menu buttons inside of the following div in order to push'em into hamburger menu when the window is shrinked,

```
<div class="collapse navbar-collapse" id="my-menus">
  <ul class="nav navbar-nav">
    <li><a href="#">About</a></li>
    <li><a href="#">Contact</a></li>
  </ul>
  <ul class="nav navbar-nav navbar-right">
    <li><a href="#">Sign Up</a></li>
    <li><a href="#">Login</a></li>
  </ul>
</div>
```

-** The hamburger button,

```
   <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
data-target="#my-menus" aria-expanded="false">
       <span class="sr-only">Toggle navigation</span>
       <span class="icon-bar"></span>
       <span class="icon-bar"></span>
       <span class="icon-bar"></span>
   </button>
```

- In the Hamburger button**, 'data-target' ATR must match the 'id' ATR in the collapse div*!

- The header section of the navbar,

```
<div class="navbar-header"></div>
```

Brand and Hamburger menu button are placed inside of this header part.

- The left side menu tags of the navbar,

```
<ul class="nav navbar-nav"></ul>
```

- The right side menu tags of the navbar,

```
<ul class="nav navbar-nav navbar-right"></ul>
```

The Grid System

Maybe it's the main reason that people prefer using bootstrap! As I shrink the window, apart from the content being scaled, it adapts into mobile or tablet mode easily and visually looking awesome,

https://getbootstrap.com/docs/3.3/css/#grid

https://getbootstrap.com/docs/4.5/layout/grid/

- It scales up to 12 columns. Anytime we use the grid, we must include it into a container! It consists of row and column divs. And so an example usage of grid system is as following,

```
<div class="container">
  <div class="row">
    <div class="col-lg-6 blue">COL LG 6</div>
    <div class="col-lg-6 blue">COL LG 6</div>
  </div>
</div>
```

- There are 4 sizes to be used in the Grid System:

1. xs: For mobile size
2. sm: For tablet size
3. md: For small screens or windows (laptop,PC)
4. lg: For big screens and windows

We can specify portions for each of these sizes.

- If we don't specify the lg size explicitly, it will take it form the md element,

```
<div class="col-md-3 col-sm-6 blue">TOUR DATE!</div>
<div class="col-md-3 col-sm-6 blue">TOUR DATE!</div>
<div class="col-md-3 col-sm-6 blue">TOUR DATE!</div>
<div class="col-md-3 col-sm-6 blue">TOUR DATE!</div>
```

So here when it's lg size, 4 of them will be on the same line as in md; when it's sm size they will get seperated 2 by 2; and finally when it's xs they will go 1 by 1 because we didn't specify it and when we don't specify a smaller size or any size, it will take full 12 columns which means 1 element at 1 line.

- Btw above the number coming after the size specifies how many columns the element will be on,

```
<div class="col-lg-6 blue">COL LG 6</div>
<div class="col-lg-6 blue">COL LG 6</div>
```

Here it will be on 6 columns, which means there will be 2 of'em on the line when the size is lg

- We can make nested grids.

- `<nav class="navbar navbar-inverse"></nav>`
    Gives us a reversed navbar (with a deep background and light text by default)

- `<div class="thumbnail"></div>`
    Creates a standart box around the element inside of it which is generally an image. If we don't use it around an image even in a grid, image won't fit proplerly and will go as its size!

- Ctrl + D: Selects all the instances of a selected text one by one on each push within Sublime Text 3, which will implicitly duplicate the cursor. It's better to choose the text from beginning to end by letting the cursor be at the end not at on the front!

- If we want to get rid of the changing size images -especially hight; because when we apply thumbnail width is fixed but hight is not- then we can see the code here,
    https://codepen.io/nax3t/pen/MJwpdb

- To solve the white space bug along the image places,
    This bug can be fixed by adding a class to the div with class of row, name it flex, then make a css rule like so:

```
.flex {
  display: flex;
  flex-wrap: wrap;
}
```

- Glyphicons,
    https://getbootstrap.com/docs/3.3/components/
    No Glyphicons in Bootstrap 4,
        https://www.glyphicons.com/
        https://primer.style/octicons/
    Also a nice alternative, for both 3 and 4,
        https://fontawesome.com/

- Glyphicons has the feature of text in terms of colors; so color ATR will change its color directly and so the font-size will change its size.

- To make the navbar fixed on the top while scrolling down, just add class,
    navbar-fixed-top
  After making this we will realize that our content stayed behind it which means on the top we need padding, so we will give our body 70px top padding which is the hight of the navbar by default,
    padding-top: 70px;

- It is important that we include our own style file after the line we include bootstrap, otherwise we cannot overwrite them!

- When we try to overwrite styles of elements and are not able to do, it is always the best solution to go the page and checking the most specific ATR that effects the element we want to change by 'inspecting element'

- In Navbars
  Div class 'container' strecthes the bar all along the screen but keeps the content in the container,

```
<nav class="navbar navbar-default">
  <div class="container">
  </div>
</nav>
```

  But div class 'container-fluid' keeps both the content and the bar in the container, which is the default actually,

```
<nav class="navbar navbar-default">
  <div class="container-fluid">
  </div>
</nav>
```

- `background-image: url();`
  `background-size: cover;`
  `background-position:center;`

- In order to make the background image cover all the file,

```
html {
  height: 100%;
}
```

- An awesome shadow that contributes depth to the texts on the page,

```
text-shadow: 0px 4px 3px rgba(0,0,0,0.4),
             0px 8px 13px rgba(0,0,0,0.1),
             0px 18px 23px rgba(0,0,0,0.1);
```

/*text-shadow: offset-x offset-y radius color;*/

https://developer.mozilla.org/en-US/docs/Web/CSS/text-shadow

- If we want our bootstrap styled website to be responsive on mobile then we should to add the following meta tag to our `<head>` element, above the `<title>` tag:

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

- To fix the appearance of the `<hr>` on mobile devices, add max-width: 90%; to the hr selector,

```
hr {
  width: 400px;
  max-width: 99%;
}
```

Chapter 5
Bootstrap 4

Releases:

1. 'Alpha Release' is the first release and it basically tells that the current suftware is potentially very buggy and very likely to change mostly.
2. 'Beta Release' is a more stable version and likely to be changed again with breaking changes just like in Alpha.
3. 'Stable Release' is the done enough and ready to publish version.

- The documentation that cover all the changes from Bootstrap 3 to 4:
  https://getbootstrap.com/docs/4.5/migration/

- Main changes,
  1. Flexbox is added and on by default,
  2. As primary unit 'rem' is selected over 'px'
  3. Global size is switched from 14px to 16px
  4. In the grid system '-xl' option is added
  5. 'Panels, thumbnails & wells' are changed with 'cards'
  6. No more 'Glyphicons' insted we can use 'Fontawesome' And more more changes...

- While including scripts: It is essential to put them in the following order at the end of the body,
  1. jQuery
  2. Popper.js
  3. Bootstrap JS

- The pre-prepared Starter-Template,
  https://getbootstrap.com/docs/4.5/getting-started/introduction/#starter-template

- Colors,
  https://getbootstrap.com/docs/4.5/getting-started/theming/#color
  https://getbootstrap.com/docs/4.5/getting-started/theming/#theme-colors
  https://getbootstrap.com/docs/4.5/utilities/colors/#background-color
  Text Colors,
  https://getbootstrap.com/docs/4.5/utilities/colors/
  Colors are brighter now in Bootstrap 4 than 3 and also it has bg-color, bg-light etc.

Typography between Bootstrap 3 and 4:
https://getbootstrap.com/docs/4.5/content/typography/
At this point there is a class type for heading called as 'display heading',
https://getbootstrap.com/docs/4.5/content/typography/#display-headings
Blockquotes,
https://getbootstrap.com/docs/4.5/content/typography/#blockquotes

```
<blockquote class="blockquote">
  <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Integer posuere erat a ante.</p>
</blockquote>
```

If we don't want the styling here we simply don't include the class="quote" part. Here class="mb-0" just sets the margin bottom of the paragraph to 0. And if we want the blockquote to be aligned right, we can add class "text-right" to the blockquote. But back in Bootstrap 3 there was a class called as "blockquote-reverse" for this mission,

```
  <blockquote class="blockquote text-right">
```

Citation,

```
  <blockquote class="blockquote">
    <p class="mb-0">Lorem ipsum dolor sit amet, consectetur adipiscing
elit. Integer posuere erat a ante.</p>
    <footer class="blockquote-footer">Someone famous in <cite
title="Source Title">Source Title</cite></footer>
  </blockquote>
```

title attribute usually refers to the title of a work you are citing not the author

- It is now enough to change the default font size to make every text on the page to change dynamically. That's because they set the main unit size as rems not as pxs like back in Bootstrap 3.

Some features that didn't exist in Bootstrap 3 at all,
Borders, (Bordered, Rounded, Left Blank, Color)
`border | rounded | rounded-top | border-left-0 | border-success`
https://getbootstrap.com/docs/4.5/utilities/borders/
Spacing, (m & p t/b/l/r/x/y - 0/1/2/3/4/5/auto)
`p-x-3` -> make padding size-3 from both sides
https://getbootstrap.com/docs/4.5/utilities/spacing/
Responsive Breakpoints, (responsive version of spacing for devices)
`p-md-5` -> start making padding 5 starting from size m up to xl
PS: We don't specify the xs size explicitly, we use default to specify that size and we use the other sizes to move on like by specifying sm and furthermore...
https://getbootstrap.com/docs/4.5/utilities/spacing/#notation
Breakpoints,
https://getbootstrap.com/docs/4.5/layout/overview/#responsive-breakpoints
`xs` - Portrait Mode Smartphones
`sm` - Landcape Mode Smartphones
`md` - Tablets
`lg` - Laptop Computers
`xl` - Desktop Computers

Navbars: (https://getbootstrap.com/docs/4.5/components/navbar/)
- We now have to specify the time of collapse of the menu (hamburger menu) in terms of breakpoints (sm-md...)
- Also the ".bg-*" class is required in Bootstrap 4
- "navbar-default" is changed to either "navbar-light" or "navbar-dark"
- If I just change the navbar to "navbar dark", just the text will be changed to white, in order to do it in real means, I also need to change the background to "bg-dark" Here I have the built-in options that I used before both at "text" and "btn" which are `"info-warning-success-primary-secondary-danger"`
- `"navbar-expand-{breakpoint}"` specifies the size that the navbar will be extended

- When we type i.e. "Lorem60" and hit Tab, it will give us 60 words long paragraph.

Display: (https://getbootstrap.com/docs/4.5/utilities/display/)
- Instead of "visible" and "hidden" in Bootstrap 3, we now have ".d" class,
```
d-{value} for xs
 d-{breakpoint}-{value}
 <h1 class="d-none d-lg-block d-xl-none">Always hidden except LG</h1>,
```
means, display none starting -including- from the breakpoint "xs" -not explicit because we don't use the breakpoint name explicitly when it's xs- and when came to XL then display as block.

- Typing i.e.
button.btn.btn-info.btn-lg
creates the following,
```
<button class="btn btn-info btn-lg"></button>
```
But if we don't specify which tag it's gonna be like by not writing "button" at the beginning, then it will create a "div" tag no matter what the classes are. i.e.
```
.border.border-dark
```
creates,
```
<div class="border border-dark"></div>
```

Flexbox (https://getbootstrap.com/docs/4.5/utilities/flex/)
- Flexbox includes a bunch of CSS properties to move things around the position elements inside of a container/page. Lots of Bootstrap 4 utilities have a usage with "flex". But actually flex is a world that exists outside of Bootstrap, by itself.

- There are two directions in the Flexbox that we need to know about. The first, by default our "main axis" of our content, is from left to right as from "start" to "end". And also the "cross axis" is from top to bottom as from "start" to "end", by default.

- There is a property in Bootstrap that lets us change how items are distributed accross the main axis. Also we can also change the direction of our flexbox which means letting it go from right to left or bottom to top where things go a bit crazy. So by default,
"Flex direction is left to right & top to bottom."

- `"justify-content"` property is the one that allows us to move the items and the default value is, - `"justify-content-start"` which -by default again- puts everything to the left. In this case if I say,
- `"justify-content-end"` it will push the elements to the right. Furthermore,
- `"justify-content-center"` centers the elements
- `"justify-content-between"` distributes all the elements evenly spaced accross the flexbox by sticking the edge elements to left and right. But,
- `"justify-content-around"` will distribute the elements evenly spaced again but this time also making a space at the edges.

- By default elements inside the flexbox are stretched from top to bottom. This happens because by default it is
- `"align-items-stretch"`. But we can use following ones to change it and align items vertically,
- `"align-items-start"`
- `"align-items-center"`
- `"align-items-end"`
- `"align-items-baseline"`

Above we can also add Breakpoints -in other words, we can use them responsively-. Like,

```
justify-content-{breakpoint}-around
  align-items-{breakpoint}-center
```

Flex Direction (https://getbootstrap.com/docs/4.5/utilities/flex/#direction):
- By default the direction ATR is `"flex-row"` which makes the direction left-to-right; if we change it to, `"flex-row-reverse"` the direction will be changed to, right-to-left.

- If we add "flex-column" ATR; things will go from top to bottom as accepting the y-axis as the main axis. So if I use "justify-content" ATR with this ATR added, it will change the placement not on x-axis but on y-axis, which means vertically. One more thing; now "align-items" is effecting not the y-axis but x-axis as expected because with the "flex-column" ATR added we have changed the main-axis to y; and cross-axis to x so "align-items" will now be effecting our latest cross-axis which is x. Same rules apply to "flex-column-reverse" by the way.

- We can use direction with breakpoints,

```
"flex-md-column"
  "flex-lg-row"
```

Navs (https://getbootstrap.com/docs/4.5/components/navs/)
- `"ul > li > a"` structure is not enforced anymore. `"nav > a"` is enough
- Navs are rewritten in Bootstrap 4 with Flex which means `"d-flex"` is included implicitly by default at the class at the beginning (rather `"ul"` or `"nav"`). So I can manipulate things by using Flex ATRs above.

The Grid System (https://getbootstrap.com/docs/4.5/layout/grid/)
- Flex is enabled!
- There are slight differences/similarities from/as the one back in Bootstrap 3:
1. Sizes are shifted back for 1 field which means xs went from ~768 to ~576 and the other shifted one back and now we have `"xl"` as an addititon
2. Of course we don't write `"xs"` here also explicitly (we had to back in BS3)
3. It must be in a "container" or a "conatiner-fluid" (which goes all the way on the screen), then we have `"row"` divs and under those we have `"col"` elements (divs,h1s etc.) as many as we wish.

4. Still 12 units
5. If I make it responsive (breakpoint based) things will be stacked at each other's top vertically at xs (! May need to be verified!)
6. If I don't specify the unit along the elements they will automatically share the space evenly. If I just give one a specific size so the rest will share the remaining space (units).

- `background: url("imgs/header.jpeg") center center / cover no-repeat;`
The line above, is a short version of those long ATRs, it centers the image and makes it cover and sets to no repeat.

Cards (https://getbootstrap.com/docs/4.5/components/card/)
- Flex is included

- `section > container > row > col > card`

- `"btn-outline"` is a nice looking button (like button etc.)

- If I want specific items to be aligned specifically within a flexbox, I can use,
`"align-self-{start/end/center}"`
Also I can use it with breakpoints,
`"align-self-{breakpoint}-{start/end/center}"`

- `"navbar-expand-md"` Extends the Navbar at MD, so at XS and SM it is collapsed

- `"fixed-top"` fixes the navbar to the top as the page is scrolled

- `"img-fluid"` makes the image responsive and dynamic especially when using it in a grid

- `"@media (max-width: 1050px) {}"` changes the things inside of it at a specific size. Called as "Media Query"

- `class="order-{breakpoint}-1"`, gives that specific element an order,
`class="order-md-2"`

- `"transition: background 500ms;"`
When the background is changed, specifies its duration to change.

Chapter 6
Introduction to JavaScript

- To open JS developer console on Chrome,
  RightClick -> InspectElement -> "Console"Tab
  It is used to play around and test things out, basically.

- Five primitive datatypes of JS:
  1. Numbers (4,9.3,-3): It is a category for all types of numbers...
  2. Strings ("Hello!",'43'): Single/Double quotes, both OK.
  3. Booleans (true,false)
  4. "null" and "undefined"

- Classical operations with numbers:
  `+,-,/,*,%`

- Operations with Strings:
  1. Concatenation:
     `"ali" + "veli"` *//"aliveli"*
  2. Escape character is \
     \\,\"
  3. "Length" property:
     `"ali".length` *//11*
  4. Accessing individual elements: (indexing starts with 0)
     `"ali"[1]` *//a*

  Ex: `"ali\\".length` //equals 4 because one of the backslashes is the escape character! And so won't even be showwing up in the original string.

- Three types of naming the variables:
  1. camelCase: capital letter is in lowercase but the capital letter of every following word is in upper case
  2. snake_case: all in lowercase but there is an underscore between every word
  3. kebab-case (dash-case): all in lowercase but there is a dash between every word
  "camelCase" is used in JS

Defining Variables in JS
- JS has dynamic typing which means we can change the variable types from one to another with just reassigning, it doesn't require any additional steps

- To store variables in JS, we use `"var"`, `"let"` or `"const"` keywords,
  1. var
     - var is scoped to "current execution context" AKA a variable's enclosing function or the global scope
     - can be reassigned whenever wanted
     - initializing value is optional
     - can be redeclared at any point (its datatype can be changed)
     - global variables are added to window

     Examples:

```
var name = "Omer";
   name + "Erol" = "OmerErol"
   name = 99; //reassigning the variable with a different type
     2 + name = 101
 var myNum = 52;
   8 + myNum = 60
   myNum = 8; //reassigning the variable
     8 + myNum = 16
```

2. let
- let is "block scoped"
- doesn't create property on global window object
- initializing value is optional
- can be reassigned
- cannot be redeclared in the same scope! (its datatype cannot be changed)

3. const
- const is also "block scoped"
- not immutable, but variable reference cannot change
- cannot be reassigned!
- must be initialized with value!
- doesn't create property on global window object
- cannot be redeclared in the same scope (its datatype cannot be changed)
- its value cannot be changed but if it's a list or sth its content can be updated but its reference cannot be

- undefined: Variables that were declared but not initilalized which means when we create a variable and not initialize it, it's value becomes "undefined",

```
var age;
```

- null: null is 'explicitly (on purpose and willingly) nothing',

```
var currentPlayer= "Charlie";
  currentPlayer = null; //game over
```

Commenting in JS
1. one-line commenting It is done with double slashes,
//this is a one-line comment
2. multiple-line commenting: It is done with slash and stars,
/ this is a multiple*
*line comment! */*

- clear() function clears the console when used on the browser console.

Built-in Methods in JS
1. alert: It pops up a message to the user, so it basically alerts the user,

```
alert("Congratulations!");
  alert(52);
  alert(192+8); //200
```

2. console.log: Prints something to the JS console so if someone doesn't have the console open, he naturally won't see this message,

```
console.log("this is my console!")
  console.log(9/3) //3
```

3. prompt: It lets us get input fron the user,

```
prompt("Tell us your age please:")
  var userName = prompt("Tell us your name please:");
```

- Some of scripts should go in the head, some of them in the body! If you put the script up top, it's going to run first.

- After selecting a word, hitting "Ctrl + D" will select the next matching word with the selected one. So by repeating this all the repeatings of that word will be selected simultaneously so that we can easily change them at the same time.

Chapters 7-8-9-10
JavaScript Basics
(Control Flow-Functions-Arrays-Objects)

--- CONTROL FLOW ---

Boolean Logic
- == Equal to
- === Equal value and type
ex:
```
x = 5
x == "5" //true
x === "5" //false
```
ex:
```
var y = null
y == undefined //true
y === undefined //false
```
- '==' performs "type coersion". It tries to get them to same format and then compare the value that's why int 5 equals to string 5.
- It is strongly recommended to use '===' because it is much safer!
- Following exmaples will give an idea about non-reliablity of '==',
ex:
```
true == "1"  //true
0 == false  //true
null == undefined  //true
NaN == NaN  //false
```

Logical Operators
1. `&&` - AND
2. `||` - OR
3. `!` - NOT
- Values that aren't actually true or false, are still inherently 'truthy' or 'falsy' when evaluated in a boolean context,
```
!"hello" //false which means string is "truthy"
!!"hell" //true which also means string is "truthy"
!!"" //false
!!null //false
!!0 //false
!!NaN //false
!!-1 //true
```
- Falsy values:
`false`, `0`, `""`, `null`, `undefined`, `NaN`

- propmt always returns a String! So in order to convert it into number,
```
var age = Number(prompt("What is your age?"));
```

Conditionals
ex:
```
if(age < 18){
    console.log("I'm sorry. You are not old enough to pass!")
  }
  else if(age < 21){
    console.log("All right, pass but my eyes are on you!")
  }
  else {
    console.log("Do whatever you want...")
  }
```

- `typeof myNumber`
  Gives us the type of the variable "myNumber"

- DRY: Don't Repeat Yourself (x WET: Write Everything Twice)
  We want to keep our code as DRY as possible.

Loops
1. while
   ex:
```
var count = 1;
    while(count <= 10) {
      console.log("count is: " + count);
      count+=2;
    }
```
When we evaluate the example above in chrome console, we will see that it also shows us number 11 which is not expected. But actually it's not printing it out it just shows us that as it is the lastly evaluated value but not printed out because at the last loop of the while count is updated to 11. And to point this out, console just pust a light-reverse-directed arrow to the beginning of the line.
   ex:
```
var str = "hello";
    var count = 0;
    while(count < str.length) {
      console.log(str[count]);
      count++;
    }
```

- `str.indexOf("a");`
  Gives me the index of first 'a' occurs in the text.
- `str.indexOf("ali");`
  Gives me the index of first 'ali' occurs in the text.
- `str.indexOf("x");`
  Gives me '-1' if x does not exist

--- FUNCTIONS ---

- `for(init; condition; step) {}`

- Functions let us wrap bits of code up into reusabel packages. They are one of the building blocks of JS,
```
funciton doSomething() { //declaring:
 console.log("Hello World!");
  }
doSomething(); //calling:
```
If I just type the function name without parantheses and hit enter, console will give me function itself but not run it!
```
doSomething
```
With arguments,
```
function area(height, width) {
      console.log(height * width);
    }
area(9, 8) //72
```

- If I leave blank an argument whilce calling a function in JS, it will take it as undefined and not cause an error

- `clear()`
  Typing and hitting enter will clear the console

- Every function in JS returns something and if we don't specify it explicitly it will simply return 'undefined'. That's the reason why we've been seeing 'undefined' consistently in the console while we were printing out console.log messages with functions

```
- function capitalize(str) {
    return str.charAt(0).toUpperCase() + str.slice(1);
  }
  var city = "wien"; //wien
 var capital = capitalize(city); //Wien
```

- `str.charAt(index)` Takes the character at 'index' of the string 'str'

- `str.toUpperCase()` Takes the string 'str' (or a single char if given) and turns it into upper case

- `str.slice(2)` Slices the string 'str' and takes the part starting (by including the given index) from the index till the end of the string

- `return` Simply stops the execution of the function

- `if(typeof(str) === "number") {}` Checks the type of str that if it's a number or not

Different Syntaxes for Creating a Function

1. Function Declaration,

```
function capitalize(str) {
     return str.charAt(0).toUpperCase() + str.slice(1);
   }
```

2. Function Expression,

```
var capitalize = function(str) {
     return str.charAt(0).toUpperCase() + str.slice(1);
   }
```

3. The slight difference between the two is, when I assign something else to the variable at the 'Function Expression' then I lose my function as expected.
4. In 'Function Expression' syntax, function names can be anonymous (can be left blank)
5. More on these two types: (https://javascriptweblog.wordpress.com/2010/07/06/function-declarations-vs-function-expressions/)

- `return num % 2 === 0;` Returns true if the num is even, false otherwise

- `str.replace(/-/g, "_");` Replaces all "-" to "_"

Scope

- It is the context that code is executed in

ex:

```
var num = 1;
  function nev() {
    var num = 2;
  }
  console.log(num); //prints 1
```

ex:

```
var num = 1;
  function nev() {
    num = 2;
  }
  console.log(num); //prints 2
```

Higher Order Functions

1. Functions that take other functions as argument.
2. `setInterval(anotherFunc, intervalMs)`: Takes another function and time as ms as arguments and repeats that function at every ms that was given. The reason why we don't use parantheses here while passing the function, we are passing the function itself as the source-code to the `setInterval()` function and lets it to call that code and execute that stuff by itself,

```
setInterval(sing, 1000);
```

3. As soon as we type and hit enter it returns us a number and then starts executing the code at given intervals. That number is the key for us to stop the interval. Let's say it was '2', so to stop the interval by the function '`clearInterval()`' we type,

```
clearInterval(2);
```

4. We also can use an anonymous -not predefined- function within the `setInterval()` like,

```
setInterval(function(){
      console.log("This is an anonymous function.")
   }, 1000)
```

5. 1000 ms = 1 second

--- ARRAYS ---

- Ways of creating an array,

1. `var friends = [];` *//common*
2. `var friends = new Array()` *//uncommon*

- Syntax of an array,

```
var friends = ["Emma", "John", "Betty", "Dave"];
```

- Index numbers in JS starts from 0,

```
friends[0]; //"Emma"
```

- We can directly change an element at an index by reassigning,

```
friends[1]; //"John"
friends[1] = "Johnny";
friends[1] //"Johnny"
```

- We can add new data to an array,

```
friends[4] = "Julia";
  friends //["Emma", "John", "Betty", "Dave", "Julia"];
```

- If we add the new data to a further place,

```
friends[7] = "Chuck";
```

Then JS will fill the indexes between with "undefined",

```
friends; //["Emma", "John", "Betty", "Dave", "Julia", undefined, undefined, "Chuck"];
```

- Arrays can hold different types of data altogether. All individual items doesn't need to be of the same type,

```
var mixed = [1, "Anna", true, undefined, 52, "Türkei", null];
```

- Length property will give us the length of the array just like strings,

```
friends.length //8
```

Built-in Array Methods (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array#)

1. `push/pop`

We use '`push`' to add item to the end of the array. It also returns the updated length of the array right away, it can be stored in a variable for further use,

```
var abc = ["a", "b"];
   var.push("c") //3
abc //["a", "b", "c"]
```

We use 'pop' to remove the last item of the array. It also returns us that last element right away, it can be stored in a variable for further use,

```
abc.pop() //"c"
abc //["a", "b"]
```

2. unshift/shift

They are similar to 'push/pop' pair but just does the same thing at the beginning of the array.

So we use 'unshift' to add item to the beginning of the array, It also returns us the updated length of the array right away, it can be stored in a variable for further use,

```
var nums = [2, 3, 4];
var.unshift(1) //4
nums //[1, 2, 3, 4]
```

We use 'shift' to remove the first item of the array. It also returns us that first element right away, it can be stored in a variable for further use,

```
nums.shift() //1
nums //[2, 3, 4]
```

3. indexOf

It's used to find the index of the first occurence of an element in an array, it simply returns us that index and if the given element is not found within the array it returns '-1',

```
var myFriends = ["Ahmet", "Mehmet", "Hasan", "Ahmet"];
myFriends.indexOf("Mehmet"); //1
myFriends.indexOf("Ahmet"); //0 (not 3)
myFriends.indexOf("Hakan"); //-1
```

4. slice

It's used to copy portions or all of the array,

```
var colors = ["blue", "red", "green", "yellow", "black"];
var myColors = colors.slice(1,3); //it copies the [1,3) elements, so first index is included but the second is not!
myColors //["red", "green"]
var colorsCopy = colors.slice(); //copies all elements
colorsCopy //["blue", "red", "green", "yellow", "black"];
```

PS: The original array is not manipulated, it stays the same

EX:

1.
```
var numbers = [11, 22, 33, 44];
console.log(numbers[numbers.legth]) //prints undefined
```
2.
```
var names = [ ["Ali", "Veli"], ["Ahmet", "Mehmet"], ["Hasan", "Hüseyin"] ];
console.log(names[1]) //["Ahmet", "Mehmet"]
console.log(names[2][0]) //"Hasan"
```

Note by Lecturer,

Chrome browser behaves a little strangely when using alert, prompt, or confirm functions. It doesn't display the HTML on the page until after the popup has been closed. This is problematic since our HTML contains instructions for the user to be able to use the app we're building.

You can avoid this by wrapping your JS code in the following setTimeout function:

```
window.setTimeout(function() {
    // put all of your JS code from the lecture here
  }, 500);
```

This gives the HTML a half second to load before running the JS, which circumvents the issue of the prompt function blocking the HTML from loading right away.

This is not something you would have to deal with in the real world as prompt, alert, and confirm functions are almost never used and when they are it's typically not on page load.

You'll also learn jQuery in latter sections which has a cool `$('document').ready()` function that you could use in place of the `window.setTimeout` workaround mentioned above.

Array Iteration

1. Classical for loop,

```
var colors = ["rot", "grün", "gelb"];
   for(var i = 0; i < colors.length; i++){
     console.log(colors[i]);
   }
```

2. `.forEach` loop,

```
arr.forEach(someFunction)
```

ex:

```
var colors = ["rot", "grün", "gelb"];
   colors.forEach(function(color){
  //color is a placeholder, it can be renamed as wished
   console.log(color);
   });
```

Here if we don't give an argument to the anonymous function inside, it will repeat the things inside the function as many times as the number of elements in the array. So in conclusion, that argument hold that specific item in the array in every iteration and gives us the chance to use it as we want. Lastly we can use either a pre-defined function or an anonymous function.

Note by Lecturer,

`.forEach` takes a callback function, that callback function is expected to have at least 1, but up to 3, arguments. This is how .forEach was designed.

The arguments are in a specific order:

- The first one represents each element in the array (per loop iteration) that `.forEach` was called on.
- The second represents the index of said element.
- The third represents the array that `.forEach` was called on (it will be the same for every iteration of the loop).

ex:

```
[1,2,3].forEach(function(el, i, arr) {
    console.log(el, i, arr);
  });
```

- Splice method is used to delete a specific item with the given index.

```
.splice(index, howManyItemsToDelete)
```

- Be careful when using `.forEach` because when we use return inside of it, we just return from the anonymous function not from the `.forEach`!

- As soon as I call the function with parantheses at the end, then the function is executed, otherwise not.

- To make myForEach function like `arr.myForEach`, we write,

```
Array.prototype.myForEach = function(func) {
  //loop through the array
  for(var i = 0; i < this.length; i++){
   //call func for each item
   func(this[i]);
   }
  }
```

--- OBJECTS ---

- It consists of key-value pairs. They are not part of an ordering.

- Objects can hold all sorts of data, nums, strings, arrays even other objects...

- An object syntax seems like the following,

```
var person = {
   name: "Cindy",
   age: 32,
   city: "Köln"
 };
```

- There are 3 ways of creating an object,

2. Create'em all at once, just like above,

```
var person = {
    name: "Travis",
    age: 22,
    city: "Linz"
  };
```

1. Make an empty object then add to it,

```
var person = {}
  person.name = "Travis";
  person.age = 22;
  person.city = "Linz";
```

3. Make the empty object with a different notation,

```
var person = new.Object();
  person.name = "Travis";
  person.age = 22;
  person.city = "Linz";
```

- Updating the properties,

```
person["age"] += 1;
 person.city = "Berlin";
```

- To retrieve data from objects there are two ways:

`1. person["name"]` *//bracket notation*
`2. person.name` *//dot notation*

- We cannot use dot notation if the property starts with a number,

`someObject.1blah` *//INVALID*
`someObject["1blah"]` *//VALID*

- We cannot use dot notation if the property has spaces in its name,

`someObject.blah blah` *//INVALID*
someObject["blah blah"] *//VALID*

- We can lookup using a variable with bracket notation,

```
var str = "name";
```
`someObject.str` *//doesn't look for "name"*
`someObject[str]` *//does evaluate str and looks for "name"*

- In some languages 'objects' are called as 'dictionaries' thanks to the key-value pairs.

- Actually arrays are a special type of objects that has the keys only as numbers.

- To add a new key-value pair to an object,

```
person.surname = "Odd";
```

- An array of object is like the following,

```
var posts = [
    {
      title: "Dogs are awesome",
      author: "Ryan",
      comments: ["Yeah I think so", "Nope!"]
    },
    {
      title: "Cats are awesome",
      author: "Bob",
      comments: ["Not really", "Of course..."]
    }
  ]
  posts[1].comments[0]; //"Not really"
  var myObject = {
    friends: [
      {name: "Ali"},
      {name: "Veli"},
      {name: "Ahmet"}
    ],
    color: "blue",
    isEvil: false
  }
  myObject.friends[0].name; //"Ali"
```

- We can add functions into the objects as well as properties and when we do it we call these functions as "methods" after this point,

```
var obj = {
  name: "Mustafa",
  age: 21,
  isCool: false,
  friends: ["merve", "osman"],
  add: function(x,y){
    return x+y;
  }
}
```

To call methods,

```
obj.add(5,4);
```

It is just like "`console.log()`". "`console`" is an object and "`log()`" is a method inside of it

- Both to avoid namespace collisions and keeping the code organized we can do sth. like following,

```
var dogSpace = {};
  dogSpace.speak = function(){
    return "WOOF!";
  }
  var catSpace = {};
  catSpace.speak = function(){
    return "MEOW!";
  }
  dogSpace.speak(); //"WOOF!"
  catSpace.speak(); //"MEOW!"
```

This is used i.e. with delete functions for posts or comments or users in real world. They are both delete functions but refer to the different objects and do slightly different tasks.

This is also used in libraries. When we import a library, simply to use its methods we first refer to the library itself which is actually an object or in other words a namespace.

- Keyword "this" is used within the methods actually,

```
  var comments = {};
  comments.data = ["Good job!", "Thanks", "Congrats."];
  comments.print = function(){
    this.data.forEach(function(el){ //here "this" refers to the "comments" object!
   console.log(el);
    });
  };
  comments.print(); //Good job! \n Thanks \n Congrats.
```